GPU TECHNOLOGY CONFERENCE

April 4-7, 2016 | Silicon Valley

NVIDIA VISIONWORKS™ PROGRAMMING TUTORIAL

Thierry Lepley, April 4th 2016

PRESENTED BY NVIDIA

TUTORIAL GOAL

Intermediate Tutorial for Developers

Understand philosophy of the API

Understand main features of the API

Start developing with VisionWorks

Extra Credit

Come and ask more questions at the VisionWorks hangout (H6115)

INTRODUCTION

NVIDIA VISIONWORKS

VISIONWORKS API

What It Gives Access To ?

Core data objects: images, arrays, pyramids, etc.

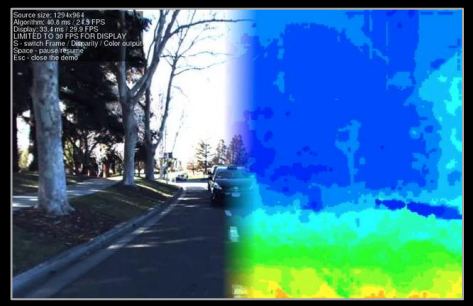
Execution Framework : graphs, nodes, delays, etc.

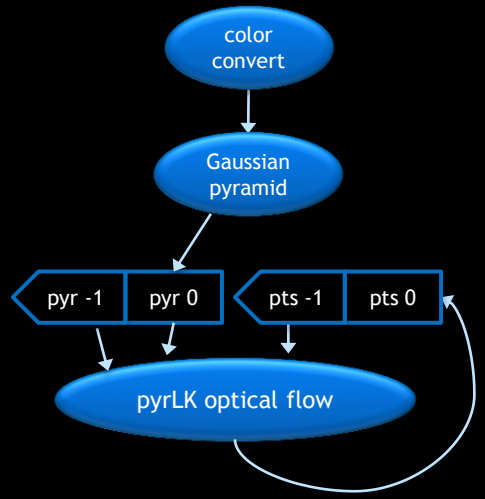
Computer Vision primitives

- Image filtering functions
- Image arithmetic and analysis
- Geometric transformations
- Feature extraction and tracking
- Depth and Flow

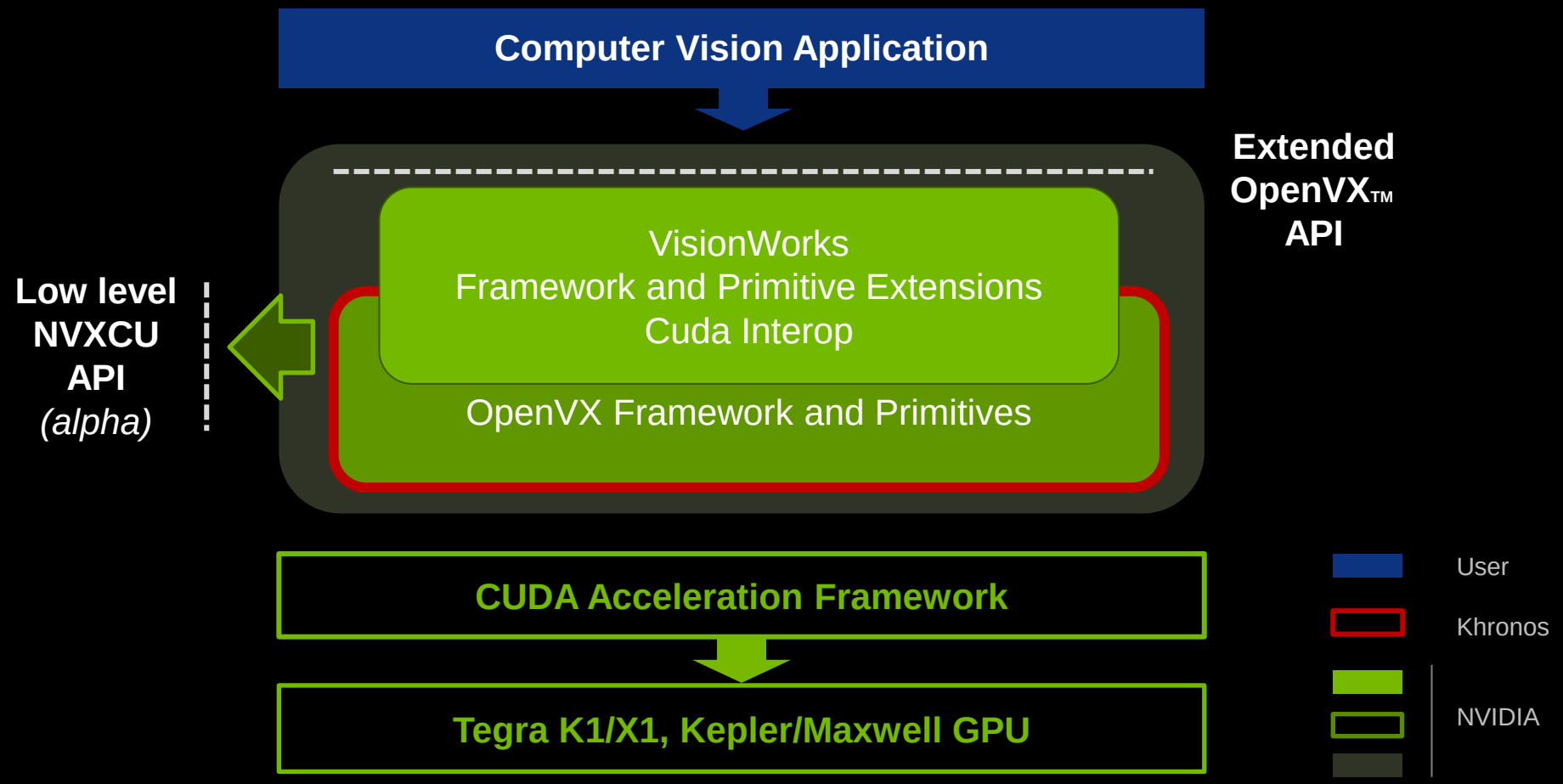
User extensibility : user kernels

CUDA Interop





VISIONWORKS SOFTWARE STACK



OpenVX Overview

KHRONOS GROUP Open consortium creating royalty-free, open standard

Main OpenVX goals

1. Define a subset of relevant primitives and image/data format
2. Enable acceleration on modern heterogeneous architectures
3. Provide portability and target performance portability across systems

NVIDIA VISIONWORKS

OpenVX Timeline

Early 2012
OpenVX Working group formed

October 2014
OpenVX 1.0 released

Jan 2015
First conformant implementation

NVIDIA VISIONWORKS

June 2015
OpenVX 1.0.1 released

Nov 2015
First public implementation

NVIDIA VISIONWORKS

AGENDA

Programming Basics

Efficient IO

Graph and Delay

AGENDA

Programming Basics

- General Philosophy
- Primitives
- Data Objects
- Code Example

AGENDA

Programming Basics

- General Philosophy
- Primitives
- Data Objects
- Code Example

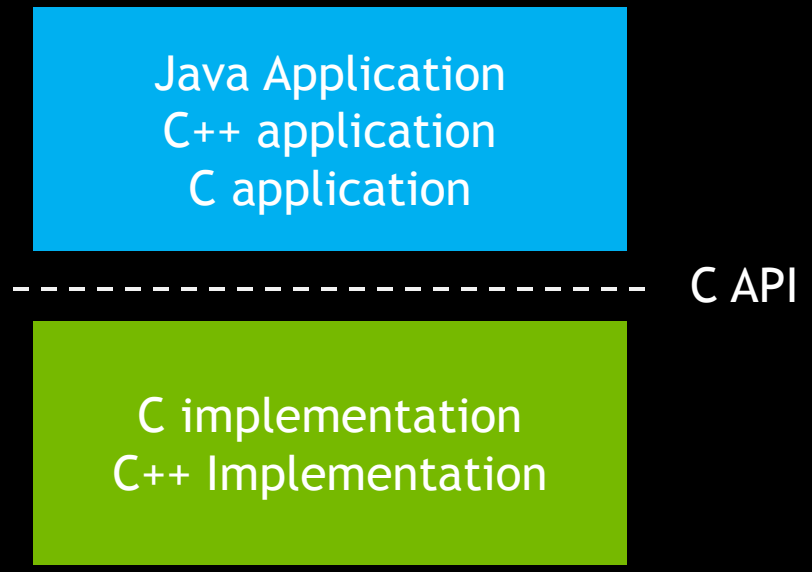
VISIONWORKS : C API

Can interop with any language

Application

Implementation of the API

No portability issue across compilers



CONTEXT

An OpenVX World

Need to be created first

```
vx_context context = vxCreateContext();
```

Objects are created in a context

```
vx_image img = vxCreateImage(context, 640, 480, VX_DF_IMAGE_RGB);
```

OBJECTS

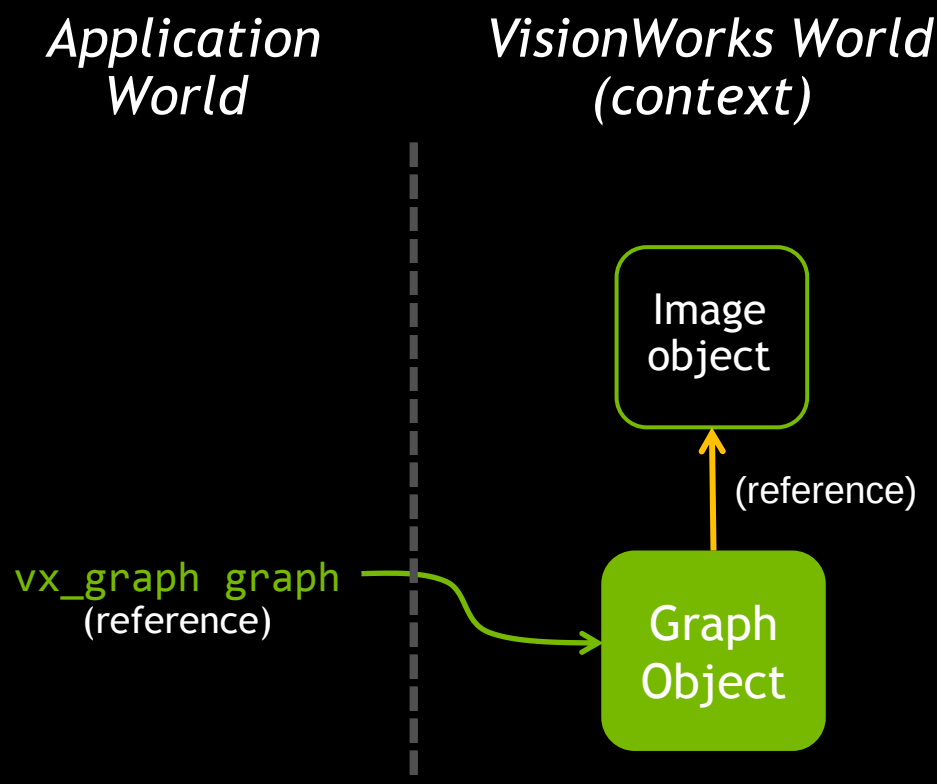
Reference Counted

```
vx_image img = vxCreateImage(context, ...);

vx_graph graph = vxCreateGraph(context);
vxBox3x3Node(graph, img, out);

vxReleaseImage(&img);
```

The Application gets object references

Object not destroyed until *ref_count == 0*

Safe Memory Management



OBJECT REFERENCES

vx_reference

One reference type per object type : vx_image, vx_array, etc.

```
vx_array array = vxCreateArray(context, ...);
vx_image img = vxCreateImage(array, ...); // Compile time error
```

Some functions work on any object reference : down-cast to vx_reference

```
vx_status status = vxGetStatus((vx_reference)array);
```

```
vxSetParameterByIndex(node, 0, (vx_reference)input_image);
```

ERROR MANAGEMENT

Status Code

Most of API calls : a vx_status code returned

```
if (vxuColorConvert(context, input, output) != VX_SUCCESS) { /* Error */ }
```

Object creation : use vxGetStatus to check the object

```
vx_image img = vxCreateImage(context, 640, 480, VX_DF_IMAGE_RGB);
if (vxGetStatus((vx_reference)img) != VX_SUCCESS) { /* Error */ }
```

ERROR MANAGEMENT

Textual Information : Log Callback

Registered in a context

Called each time an error occurs

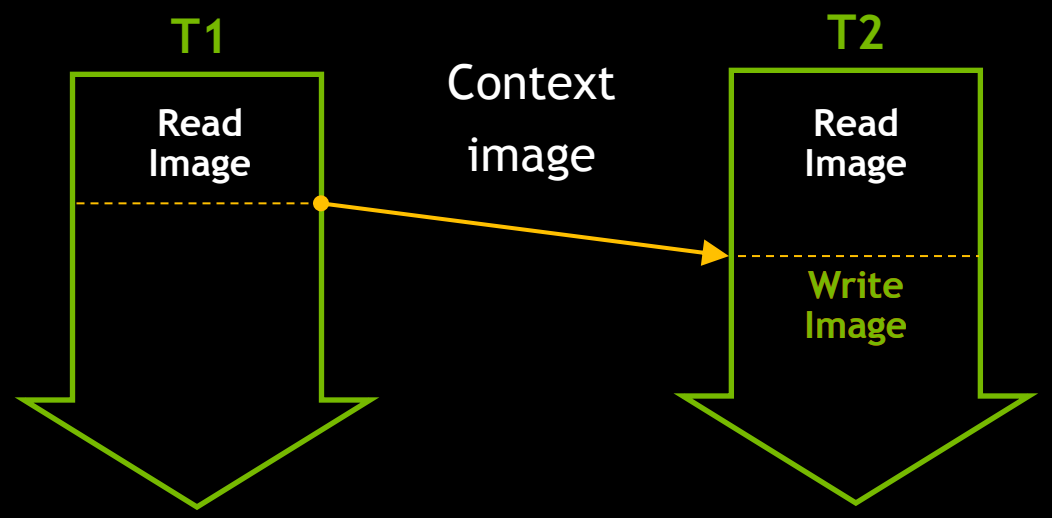
```
void logCallback(vx_context c, vx_reference r, vx_status s,
                 const vx_char string[] m)
{
  /* Do something */
}

vxRegisterLogCallback(context, logCallback, vx_false_e);
```

THREAD SAFETY

Functions: Same API function can be concurrently called from multiple thread

Objects: A context and its objects can be shared across threads

The application must ensure there is no ‘data race’ (e.g. with synchro)



ANY QUESTION SO FAR ?

AGENDA

Programming Basics

- General Philosophy
- Primitives
- Data Objects
- Code Example

NVIDIA VISIONWORKS

COMPUTER VISION PRIMITIVES

IMAGE ARITHMETIC
Absolute Difference
Accumulate Image
Accumulate Squared
Accumulate Weighted
Add / Subtract/ Multiply +
Channel Combine
Channel Extract
Color Convert +
CopyImage
Convert Depth
Magnitude
Not / Or / And / Xor
Phase
Table Lookup
Threshold

FLOW & DEPTH
Median Flow
Optical Flow (LK) +
Semi-Global Matching
Stereo Block Matching
IME Create Motion Field
IME Refine Motion Field
IME Partition Motion Field

GEOMETRIC TRANSFORMS
Warp Affine +
Warp Perspective +
Flip Image
Remap
Scale Image +

FILTERS
BoxFilter
Convolution
Dilation Filter
Erosion Filter
Gaussian Filter
Gaussian Pyramid
Laplacian 3x3
Median 3x3
Scharr 3x3
Sobel 3x3

FEATURES
Canny Edge Detector
Fast Corners +
Fast Track
Harris Corners +
Harris Track
Hough Circles
Hough Lines

ANALYSIS
Histogram
Histogram Equalization
Integral Image
Mean Std Deviation
Min Max Locations

+ *Standard with NVIDIA Extensions*
NVIDIA Proprietary

PRIMITIVES EXECUTION

2 options

Primitive → Immediate mode

Primitive → Graph mode

PRIMITIVES EXECUTION

Immediate Mode

Blocking calls similar to OpenCV usage model

Prefixed with ‘vxu’

```
// 3x3 box filter
vxuBox3x3(context, src0, tmp);

// Absolute Difference of two images
vxuAbsDiff(context, tmp, src, dest);
```

Useful for fast prototyping

PRIMITIVES EXECUTION

Graph Mode

Workload given ahead-of-time

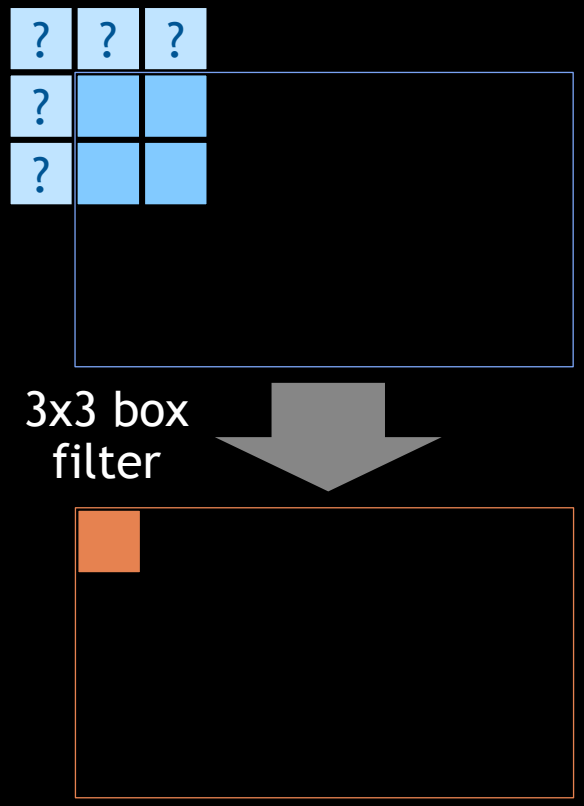
More optimization opportunities

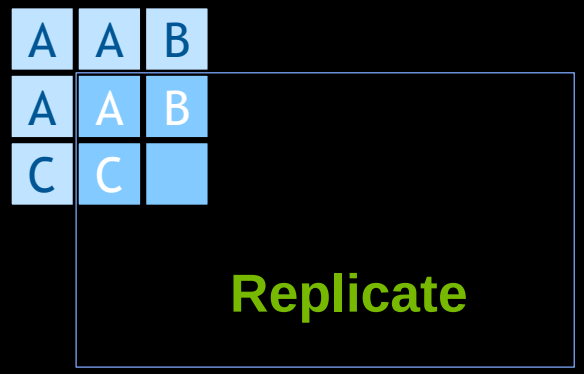
Good fit with video stream processing

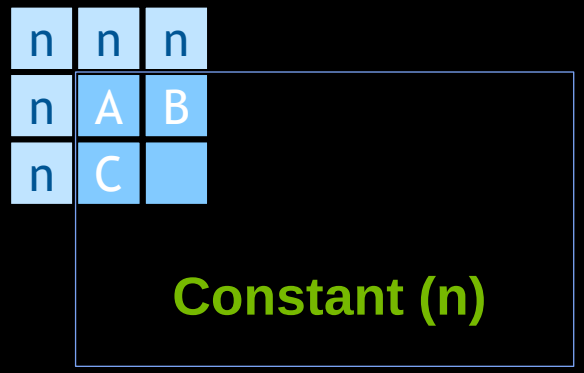
```
vx_graph graph = vxCreateGraph(context);

// Create nodes and check the graph ahead of time (errors detected here)
vxBox3x3Node(graph, src0, tmp);
vxAbsDiffNode(graph, tmp, src1, dest);
vxVerifyGraph(graph);

// Execute the graph at runtime
vxProcessGraph(graph);
```

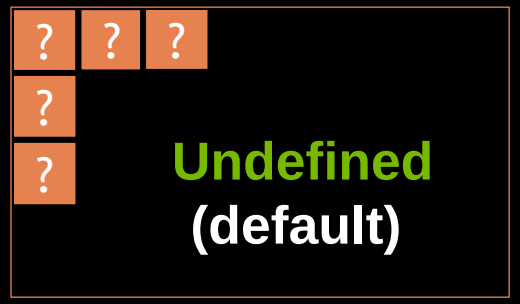
BORDER MANAGEMENT

Supported Modes









BORDER MODES

API

Enum : VX_BORDER_MODE_[UNDEFINED | CONSTANT | REPLICATE]

Immediate execution: context attribute (state)

```
vx_border_mode_t mode = { VX_BORDER_MODE_CONSTANT, 0};
vxSetContextAttribute(context, VX_CONTEXT_ATTRIBUTE_IMMEDIATE_BORDER_MODE,
                      &mode, sizeof(mode));

vxuBox3x3(context, src, dest);
```

Graph : node attribute

```
vx_border_mode_t mode = { VX_BORDER_MODE_CONSTANT, 0};
vx_node node = vxBox3x3Node(graph, src, tmp);

vxSetNodeAttribute(node, VX_NODE_ATTRIBUTE_BORDER_MODE, &mode, sizeof(mode));
```

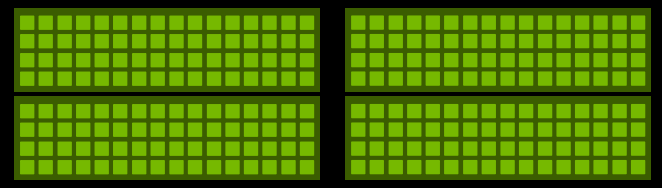
TARGET COMPUTE DEVICE

Functionality

Most primitives have both CPU and GPU implementations

GPU device ID controllable

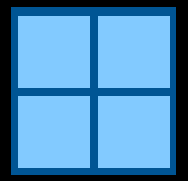
Context



GPU 1 GPU 2

Target controllable with the API
Default: automatic assignment

Primitive Execution



CPU

TARGET COMPUTE DEVICE

API

Options: NVX_DEVICE_GPU, NVX_DEVICE_CPU, NVX_DEVICE_ANY

Immediate execution: context attribute (state)

```
nvx_device_type_e target = NVX_DEVICE_GPU;
vxSetContextAttribute(context, VX_CONTEXT_ATTRIBUTE_IMMEDIATE_TARGET_DEVICE,
                      &target, sizeof(target));

vxuBox3x3(context, src, dest);
```

Graph : node setter function

```
vx_node node = vxBox3x3Node(graph, src, tmp);

nvxSetNodeTargetDevice(node, NVX_DEVICE_GPU);
```

ANY QUESTION SO FAR ?

AGENDA

Programming Basics

- General Philosophy
- Primitives
- Data Objects
  a) Data Object philosophy
  b) Focus: Images
  c) Focus: Pyramids
  d) Focus: Arrays
- Code Example

DATA OBJECTS

Images

Image: vx_image +

Image Pyramid : vx_pyramid +

Arrays

Array : vx_array +

Distribution : vx_distribution +

Look-up-table : vx_lut +

Matrices

Matrix: vx_matrix +

Convolution : vx_convolution +

Remap : vx_remap +

Scalars

Scalar : vx_scalar +

Threshold : vx_threshold +

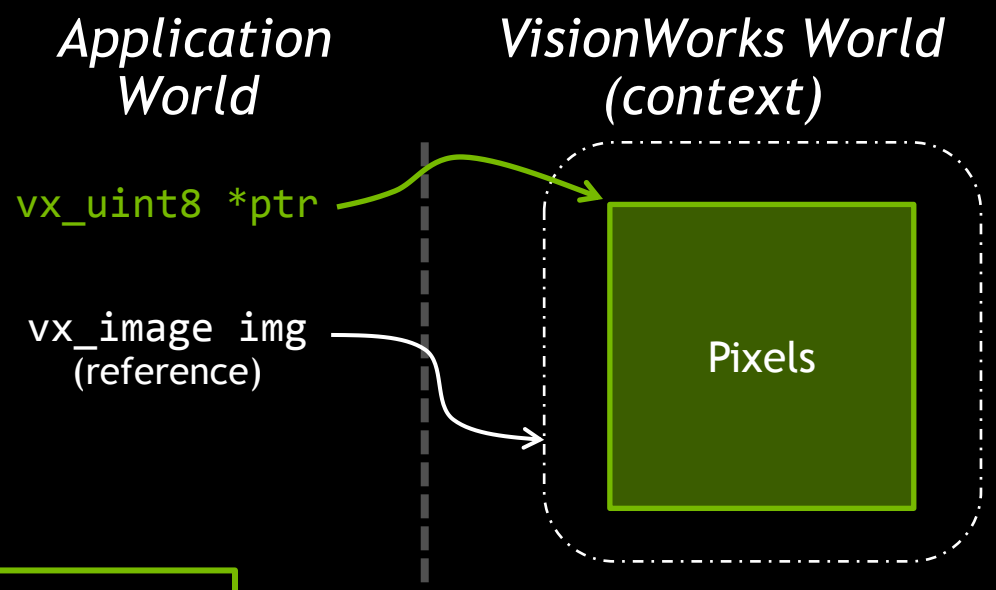
Object + Standard OpenVX with NVIDIA Extensions (ex: access from CUDA)

DATA OBJECT ACCESS

Semi-opaque Objects

No permanent pointer to data content

```
vxAccessImagePatch(img, &rect, 0, &addr, &ptr,
                   VX_READ_AND_WRITE);
// Access data at address 'ptr'
vxCommitImagePatch(img, &rect, 0, &addr, ptr);
// 'ptr' is now invalid
vxBox3x3(img, out_img);
```



VisionWorks optimizes the memory management

Synchronize data with application only when needed

Minimize data synchronization between CPU and GPU

DATA OBJECT ACCESS

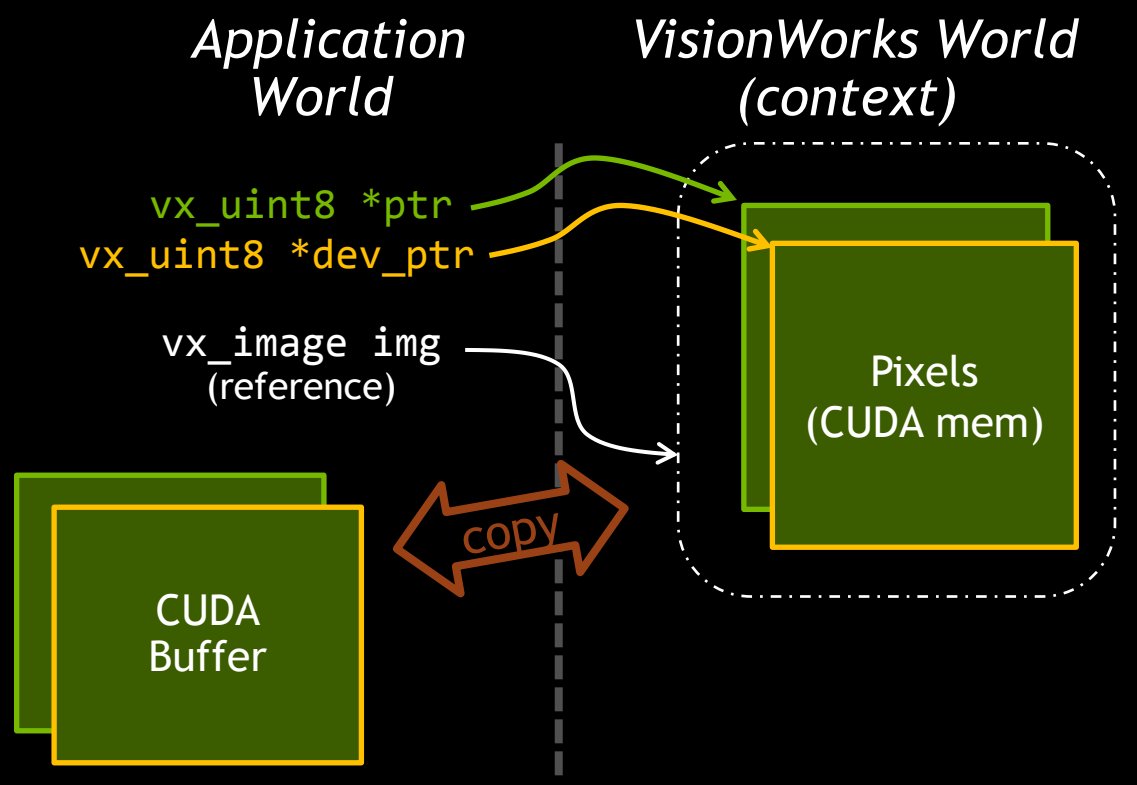
Access Modes

MAP mode (direct access)

- Host memory (CPU)
- CUDA memory

Copy Mode

- Host memory (CPU)
- CUDA memory



DATA OBJECT ACCESS

API

vxAccess<Object>(...) : access the content

HOST: VX_READ_ONLY, VX_WRITE_ONLY, VX_READ_AND_WRITE

CUDA: NVX_READ_ONLY_CUDA, NVX_WRITE_ONLY_CUDA, NVX_READ_AND_WRITE_CUDA

vxCommit<Object>(...) : release the access and commit changes

```
vx_uint8 * pLut_cu = NULL; // NULL means ‘map’, non-NULL means ‘copy’
vxAccessLUT(lut_, (void **)&pLut_cu, VX_READ_ONLY_CUDA);

// ‘pLut_cu’ is a CUDA dev pointer that can be used by CUDA kernels

vxCommitLUT(lut, pLut_cu);

// ‘pLut_cu’ is now invalid
```

FOCUS: IMAGES

Formats

Mono or Multiplanar

Color formats: RGB, RGBX, RGB16, NV12, NV21, UYVY, YUYV, IYUV, YUV4

'Gray' scale: U8, U16, S16, 2S16, U32, S32, F32, 2F32

```
vx_image img = vxCreateImage(context, 640, 480, VX_DF_IMAGE_RGB);
```

FOCUS: IMAGES

Uniform Image

Constant image

All pixels have the same value (no allocation needed)

```
vx_uint8 pix[3] = {0x0, 0x33, 0xCC};
vx_image img = vxCreateUniformImage(context, 640, 480, VX_DF_IMAGE_RGB, pix);
```

Enables performance optimizations without duplicating the primitive API

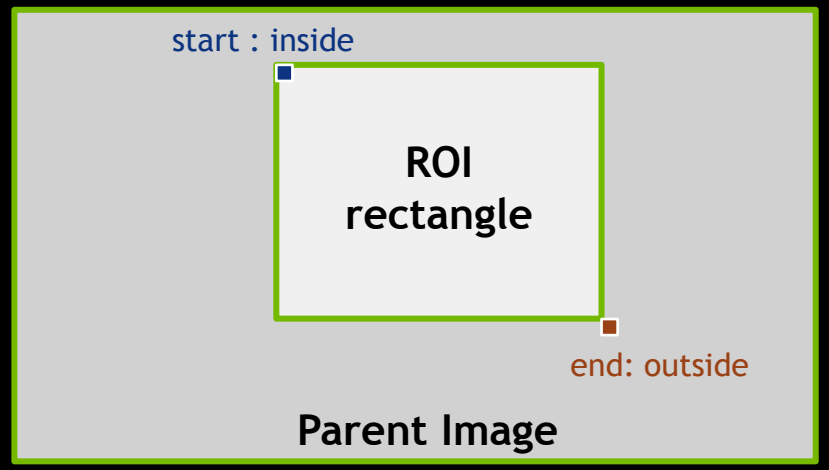
FOCUS: IMAGES

Region of Interest (ROI)

Rectangular sub-image

Same format as the parent image

Share pixels with the parent image (same memory)



struct vx_rectangle_t		
vx_uint32	start_x	The Start X coordinate.
vx_uint32	start_y	The Start Y coordinate.
vx_uint32	end_x	The End X coordinate.
vx_uint32	end_y	The End Y coordinate.

FOCUS: IMAGES

ROI Example: Stereo Images

```
vx_rectangle_t left_rect = { 0, 0, width,  height};
vx_image leftROI = vxCreateImageFromROI(inputRGB, &left_rect);

vx_rectangle_t right_rect = {width, 0, 2*width, height};
vx_image rightROI = vxCreateImageFromROI(inputRGB, &right_rect);
```

Input images from the Middlebury stereo dataset (http://vision.middlebury.edu/stereo/data)

FOCUS: IMAGES

Access

Map : VisionWorks returns *address* and *memory layout*

```
void *ptr = NULL; // NULL means 'map'
vx_imagepatch_addressing_t addr; // The memory layout will be returned here
vx_rectangle_t rect = { 0u, 0u, width, height };
vxAccessImagePatch(img, &rect, 0, &addr, &ptr, VX_READ_AND_WRITE);

// Access data at address 'ptr' with layout specified in 'addr'

vxCommitImagePatch(img, &rect, 0, &addr, ptr);
```

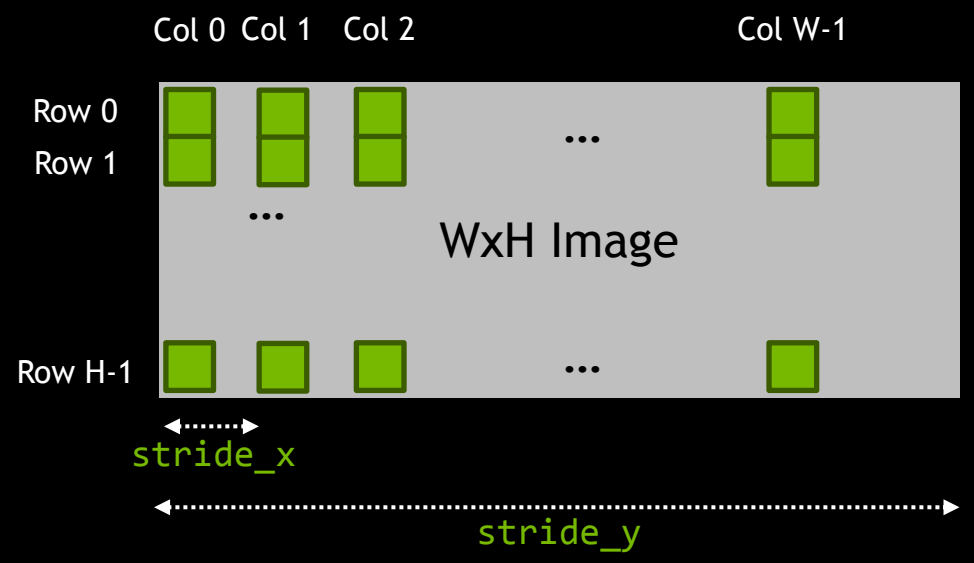
Copy : The application provides *address* and *memory layout*

```
void *ptr = &my_image[0]; // non NULL means a 'copy'
vx_imagepatch_addressing_t src_addr = { /* to fill */ };
vx_rectangle_t rect = { 0u, 0u, width, height };
vxAccessImagePatch(vx_src, &rect, 0, &addr, &ptr, VX_READ_AND_WRITE);

// Access/modify data in my_buffer

vxCommitImagePatch(vx_src, &rect, 0, &addr, ptr);
```

FOCUS: IMAGES

Memory Layout

Row Major Ordering / Pitch Linear




```
typedef struct {
    vx_uint32 dim_x;
    vx_uint32 dim_y;
    vx_int32  stride_x;
    vx_int32  stride_y;
    vx_uint32 scale_x;
    vx_uint32 scale_y;
    vx_uint32 step_x;
    vx_uint32 step_y;
} vx_imagepatch_addressing_t;
```

Nb (logical) pixels

Sub-sampled plans:
1 *physical* pixel every
'step' *logical* pixel

FOCUS: PYRAMIDS

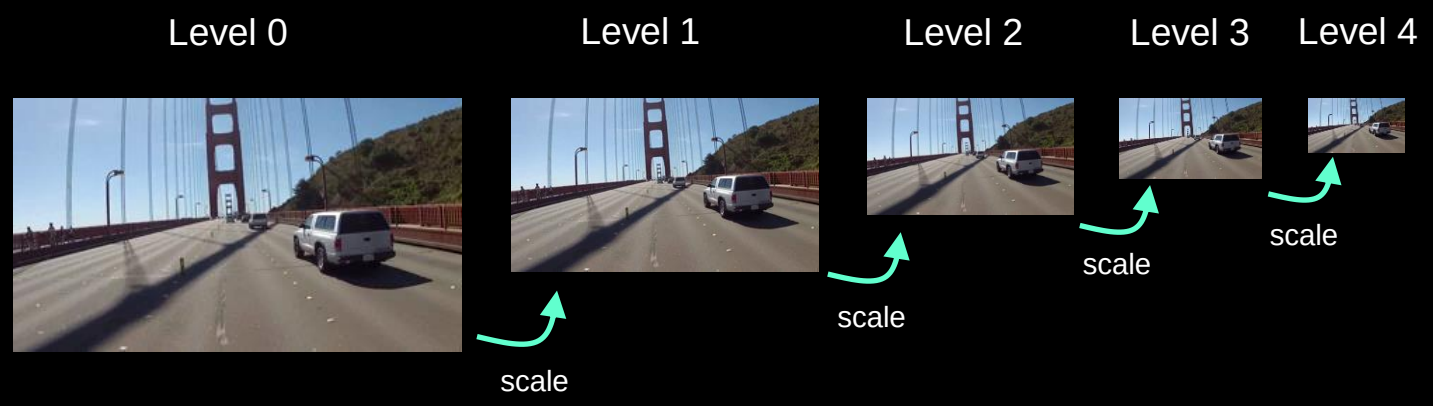
Multi-Resolution Image

Formats : same as images

Configurable number of levels

Predefined scales : VX_SCALE_PYRAMID_HALF, VX_SCALE_PYRAMID_ORB

```
vx_pyramid pyr = vxCreatePyramid(context, 5, 0.6f, 640, 480, VX_DF_IMAGE_RGB);
```



FOCUS: PYRAMIDS

More About Pyramids

Pyramid ‘levels’ are image objects

```
vx_image level1 = vxGetPyramidLevel(pyr, 1); // Increment the ref count
vxuBox3x3(context, level1, out_img);
vxReleaseImage(&level1); // Decrement de ref count
```

Pyramid used for tracking currently:

- `GaussianPyramid` : generate a pyramid from an image
- `OpticalFlowPyLK` : Lucas Kanade tracking

FOCUS: ARRAYS

Array Creation

Fix capacity (used by primitives to avoid overflow)

Variable number of items

Item types: rectangles, keypoints, coordinates 2D/3D

```
vx_array array = vxCreateArray(context, VX_TYPE_KEYPOINT, 1000);
```

A too large capacity can negatively impact the performance

FOCUS: ARRAYS

Usage Example : Dynamic ROIs

Array of rectangles for dynamic ROIs (ex: object bounding box)

```
vx_array array = vxCreateArray(context, VX_TYPE_RECTANGLE, 50);
```

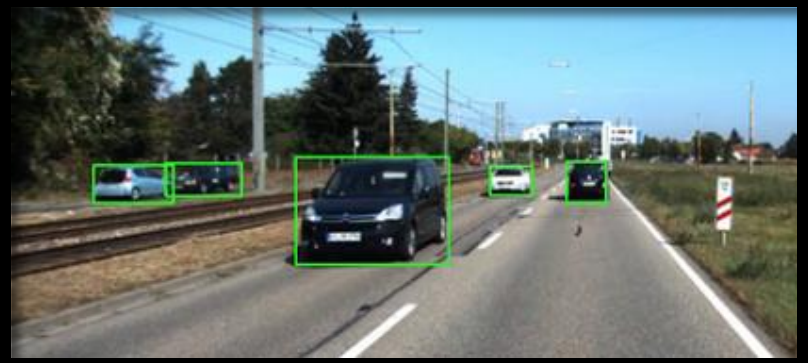


Image created from ROI for static ROI
(stereo image, ROI for static surveillance cameras)

FOCUS: ARRAYS

Access Example

Map : VisionWorks returns *address* and *stride*

```
void *base = NULL; // NULL means ‘map’
vx_size stride;
vxAccessArrayRange(array, 2, 10, &stride, &base, VX_READ_AND_WRITE);
// Access data of range [2, 10[ at address base
vxCommitArrayRange(array, 2, 10, base);
```

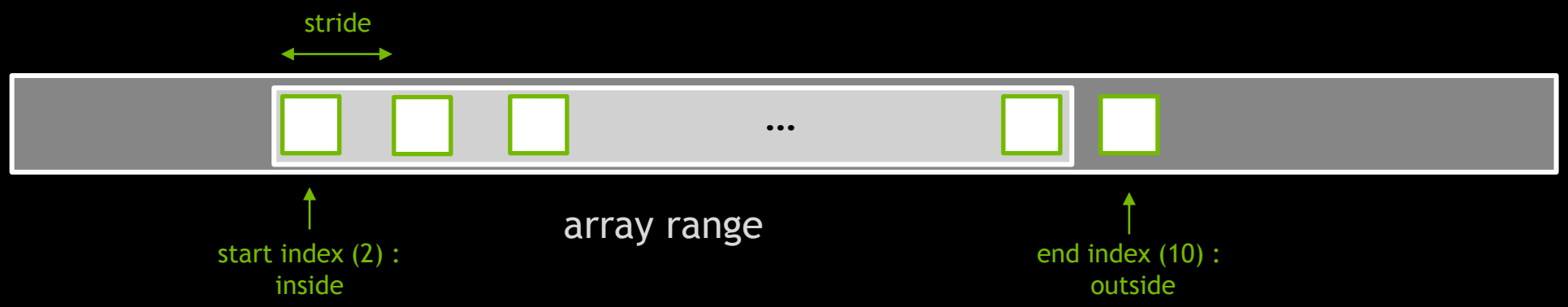
Copy : The application provides *address* and *stride*

```
void *base = &my_buffer[0]; // non NULL means a ‘copy’
void my_stride = sizeof(element_type);
vxAccessArrayRange(array, 0, 10, &stride, &base, VX_READ_AND_WRITE);
// Access data of range [2, 10[ in my_buffer
vxCommitArrayRange(array, 2, 10, base);
```

FOCUS: ARRAYS

Memory Layout

```
vxAccessArrayRange(array, 2, 10, &stride, &base, VX_READ_AND_WRITE);
```

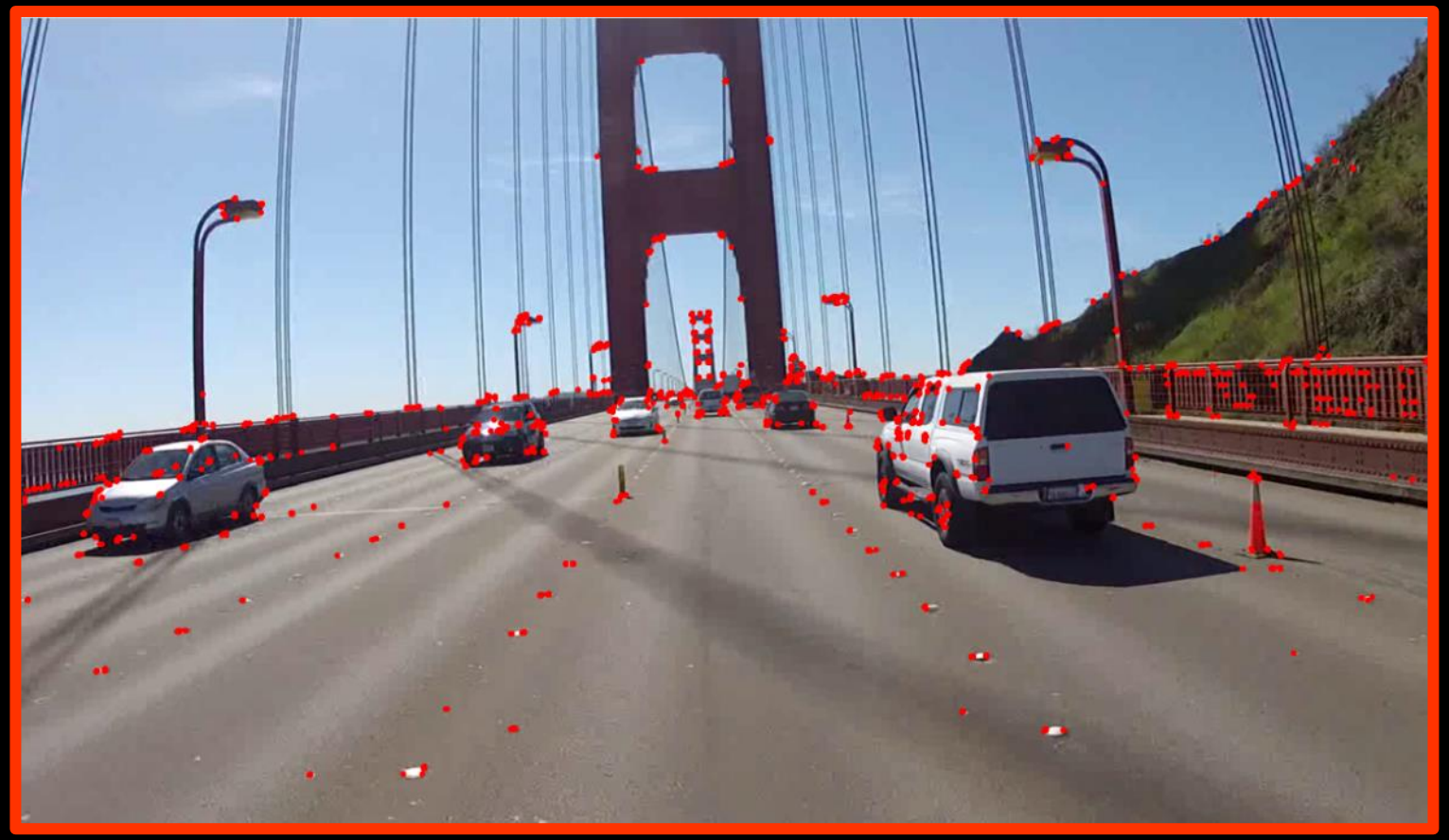


AGENDA

Programming Basics

- General Philosophy
- Primitives
- Data Objects
- Code Example

FEATURE DETECTION



FEATURE DETECTION

What is in the Toolkit ?

Feature detector primitives

- `HarrisCorner` : strongest Harris points in the image
- `FastCorner` : strongest FAST points in the image
- `HarrisTrack` : balanced (per cell) Harris (re)detection
- `FastTrack` : balanced (per cell) FAST (re)detection

Keypoint structures

- `vx_keypoint_t` : *int* coordinates, strength, tracking error & status, …
- `nvx_keypointf_t` : same as `vx_keypoint_t` except *float* coordinates
- `nvx_point2f_t` : lightweight structure, only *float* coordinates

FEATURE DETECTION

Processing

frame
(RGB image)

↓

color convert

↓

frame_gray
(U8 image)

↓

Harris corner

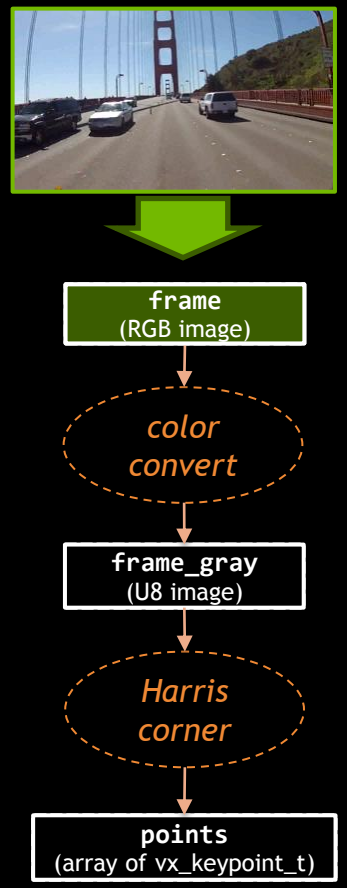
↓

points
(array of vx_keypoint_t)

FEATURE DETECTION

Prepare Data

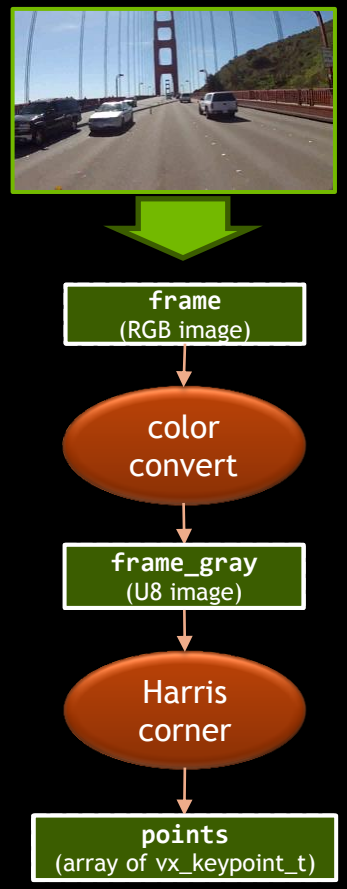
```
// RGB image data at (compact layout)
// address = pImage, width = W, height = H, compact memory layout

// Create data objects
vx_image frame = vxCreateImage(context, W, H, VX_DF_IMAGE_RGB);
vx_image frame_gray = vxCreateImage(context, W, H, VX_DF_IMAGE_U8);
vx_array points = vxCreateArray(context, VX_TYPE_KEYPOINT, 1000);

// Copy the input data into the vx_image object
vx_imagepatch_addressing_t addr;
   addr.stride_x = 3*sizeof(vx_uint8); // R + G + B
   addr.stride_y = addr.stride_x * W;
void *p = pImage; // Non NULL pointer means a 'copy'
vx_rectangle rect = {0, 0, W, H}; // Entire image
vxAccessImagePatch(frame, &rect, 0, &addr, &p, VX_WRITE_ONLY);
vxCommitImagePatch(frame, &rect, 0, &addr, p);
```



FEATURE DETECTION

Computation and Get Outputs

```
// RGB to U8 conversion
vxuColorConvert(context, frame, frame_gray);

// Keypoint detection : Harris corner
vxuHarrisCorners(context, frame_gray, s_strength_thresh, min_dist,
                  k_sensivity, gradientSize, blockSize, points, 0);

// Access keypoints
vx_size nb_kp;
vxQueryArray(points, VX_ARRAY_ATTRIBUTE_NUMITEMS, &nb_kp, sizeof(nb_kp));

vx_size stride; // Returned by the access function
vx_keypoint_t *base = NULL; // NULL means 'map' (direct access)
vxAccessArrayRange(points, 0, nb_kp, &stride, (void **)&base, VX_READ_ONLY);

// Access keypoints starting from address 'p' with 'stride'
vx_keypoint_t *p = base;
for(vx_size i = 0; i < nb_kp; i++, p = (vx_keypoint_t *)((char*)p + stride) )
  // ...

vxCommitArrayRange(points, 0, nb_kp, p);
```



STANDARD KEYPOINT

vx_keypoint_t Structure

Data Fields			Detector	Tracker
vx_int32	x	The x coordinate.	X	X
vx_int32	y	The y coordinate.	X	X
vx_float32	strength	The strength of the keypoint. Its definition is specific to the corner detector.	X	
vx_float32	scale	Initialized to 0 by (current) corner detectors.	(x)	
vx_float32	orientation	Initialized to 0 by (current) corner detectors.	(x)	
vx_int32	tracking_status	A zero indicates a lost point. Initialized to 1 by corner detectors.	X	X
vx_float32	error	A tracking method specific error. Initialized to 0 by corner detectors.		X

MORE PRECISION NEEDED ?

Simply Change the Output Array

`nvx_keypointf_t` : same as `vx_keypoint_t`, with float coordinates

```
vx_array points = vxCreateArray(context, NVX_TYPE_KEYPOINTF, 1000);
```

`nvx_point2f_t` : lightweight, no error computation

```
vx_array points = vxCreateArray(context, NVX_TYPE_POINT2F, 1000);
```

Data Fields		
vx_float32	x	The X coordinate (-1.0f when tracking lost)
vx_float32	y	The Y coordinate (-1.0f when tracking lost)

The rest of the code is unchanged

ANY QUESTION SO FAR ?

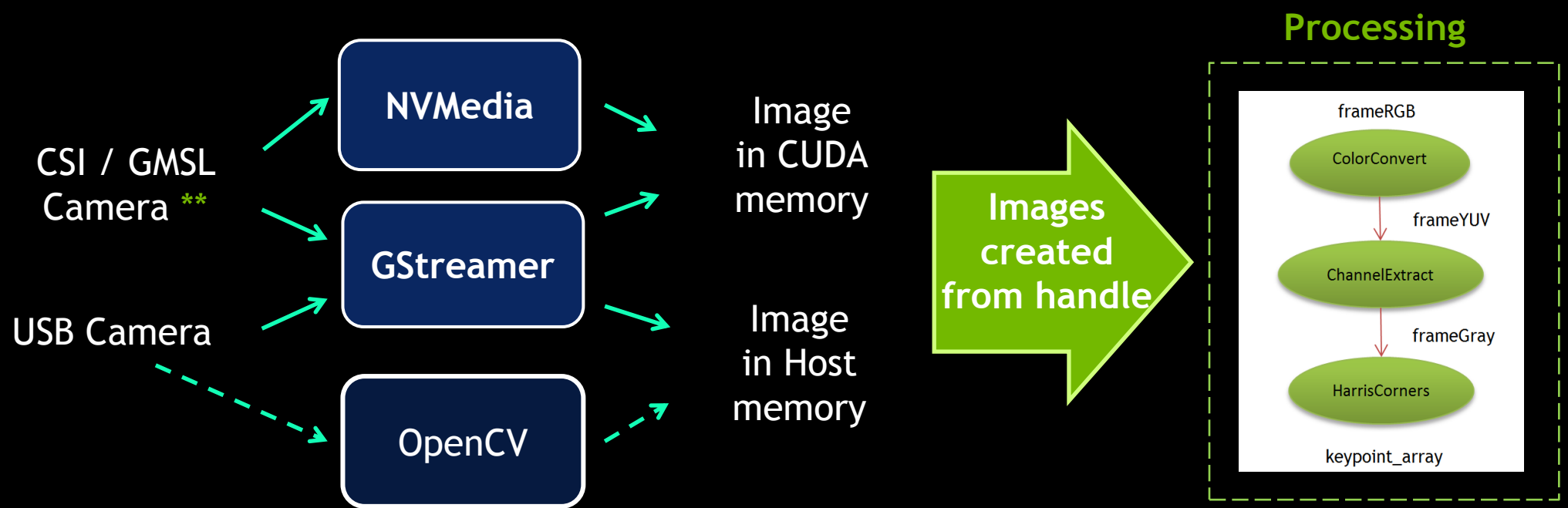
AGENDA

Programming Basics

Efficient IO

Graph and Delay

EFFICIENT IO BY AVOIDING COPIES



** NVIDIA Embedded platforms support CSI Cameras; NVIDIA Automotive platforms support GMSL Cameras

IMAGE CREATED FROM HANDLE

Principle

The handle must NOT be used directly by the application after the image object creation

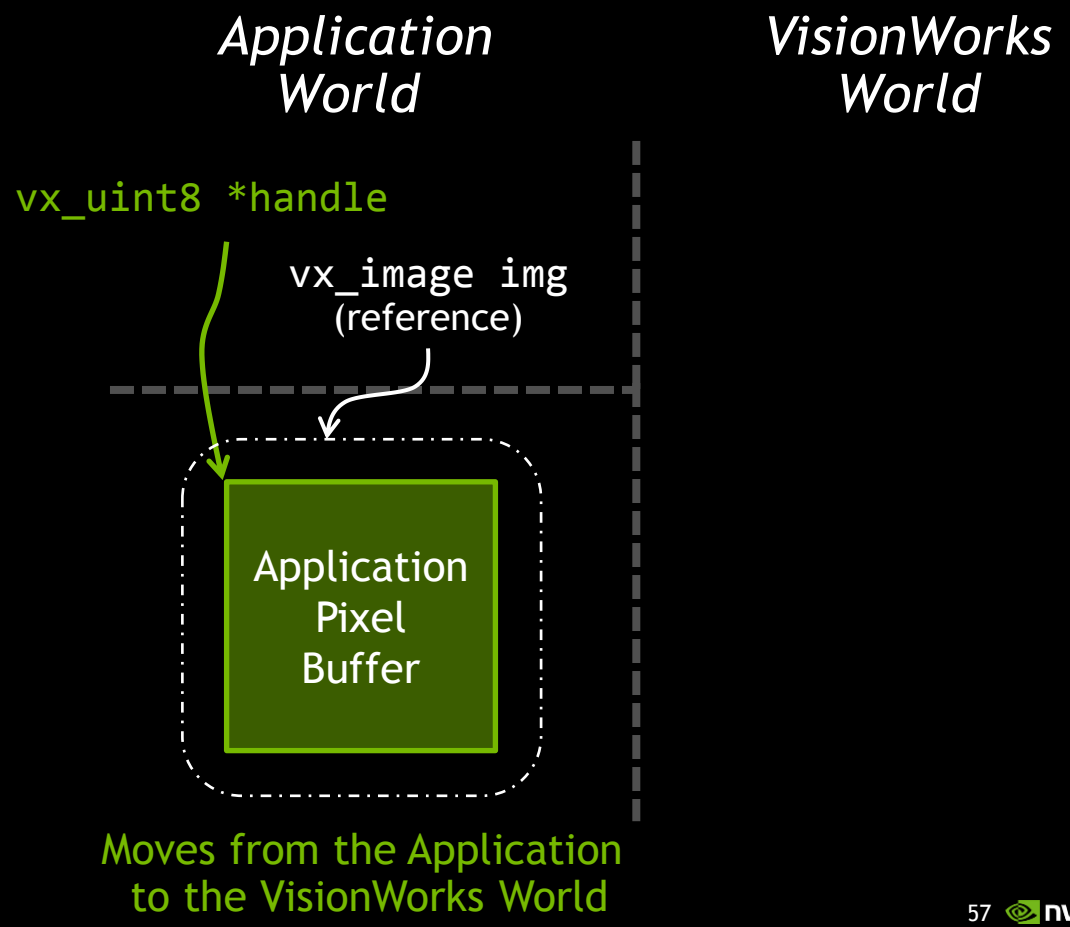


IMAGE CREATED FROM HANDLE

Creation API

CPU or CUDA memory : VX_IMPORT_TYPE_HOST, NVX_IMPORT_TYPE_CUDA

```
vx_image img = vxCreateImageFromHandle(
                        context, VX_DF_IMAGE_RGB,
                        &addr[0],  // Plane layouts
                        &ptrs[0],  // Plane handles
                        NVX_IMPORT_TYPE_HOST
               );
```

Useful for both input and output images

IMAGE CREATED FROM HANDLE

Access with Standard Access/Commit

Image access like other images : Map/Copy, Host/CUDA

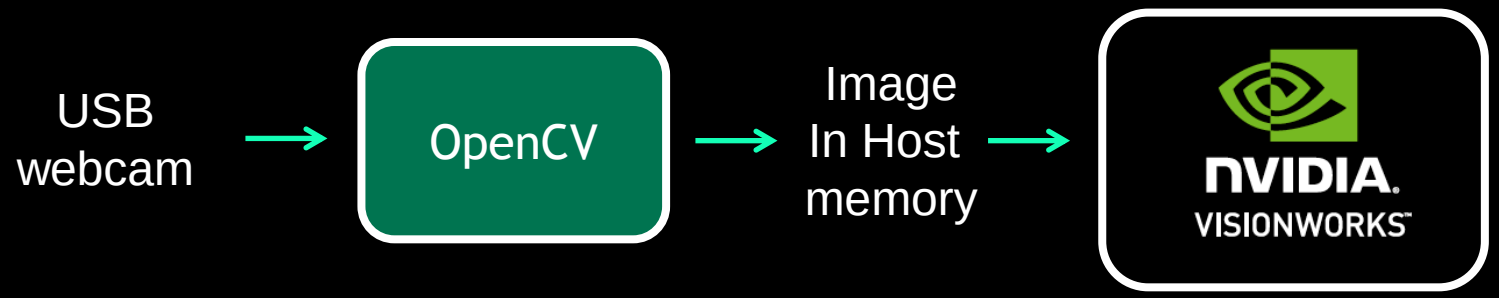
Mapped at its original address / memory layout

Property of memory back to the application at image destruction

IMAGE CREATED FROM HANDLE EXAMPLE

OpenCV Interop

Import a Webcam image into VisionWorks directly from the Host memory



OPENCV INTEROP EXAMPLE

Import a webcam image

```
// Create a Video Capture from OpenCV
cv::VideoCapture inputVideo;
inputVideo.open(0); // Grab data from the default webcam

// VideoCapture always returns a BGR image, transform it into RGB
cv::Mat cv_bgr, cv_rgb;
inputVideo.read(cv_bgr);
cv::cvtColor(cv_bgr, cv_rgb, cv::COLOR_BGR2RGB);

// Import into VisionWorks
vx_imagepatch_addressing_t addr;
  addr.dim_x = cv_rgb.cols;
  addr.dim_y = cv_rgb.rows;
  addr.stride_x = 3*sizeof(vx_uint8);
  addr.stride_y = cv_rgb.step;
void *ptrs[] = { cv_rgb.data };

vx_image img = vxCreateImageFromHandle(context,
    VX_DF_IMAGE_RGB,  &addr, ptrs, VX_IMPORT_TYPE_HOST);
```

OPENCV INTEROP EXAMPLE

Refresh an Image

```
// Mapping an image created from handle will map at the
// exact same address and with the same memory layout
void *base = NULL; // NULL means 'map'
vx_imagepatch_addressing_t addr;
vx_rectangle_t rect = { 0u, 0u, cv_rgb.cols, cv_rgb.rows};
vxAccessImagePatch(img, &rect, 0, &addr, &base, VX_WRITE_ONLY);

// Refresh the OpenCV image
inputVideo.read(cv_src_bgr);
cv::cvtColor(cv_src_bgr, cv_src_rgb, cv::COLOR_BGR2RGB);

// Commit back changes
vxCommitImagePatch(img, &rect, 0, &src_addr, base);
```

ANY QUESTION SO FAR ?

AGENDA

Programming Basics

Efficient IO

Graph and Delay

- Graphs
- Delay Object
- Code Example

AGENDA

Programming Basics

Efficient IO

Graph and Delay

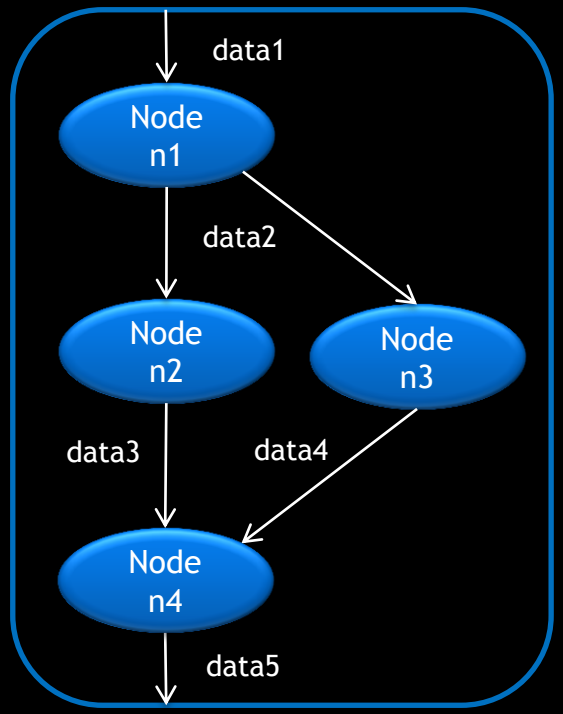
- Graphs
- Delay Object
- Code Example

WHAT IS A GRAPH ?

Computer vision pipeline specified ahead-of-time

Adapted to processing of video streams

Best for performance : enables global optimizations

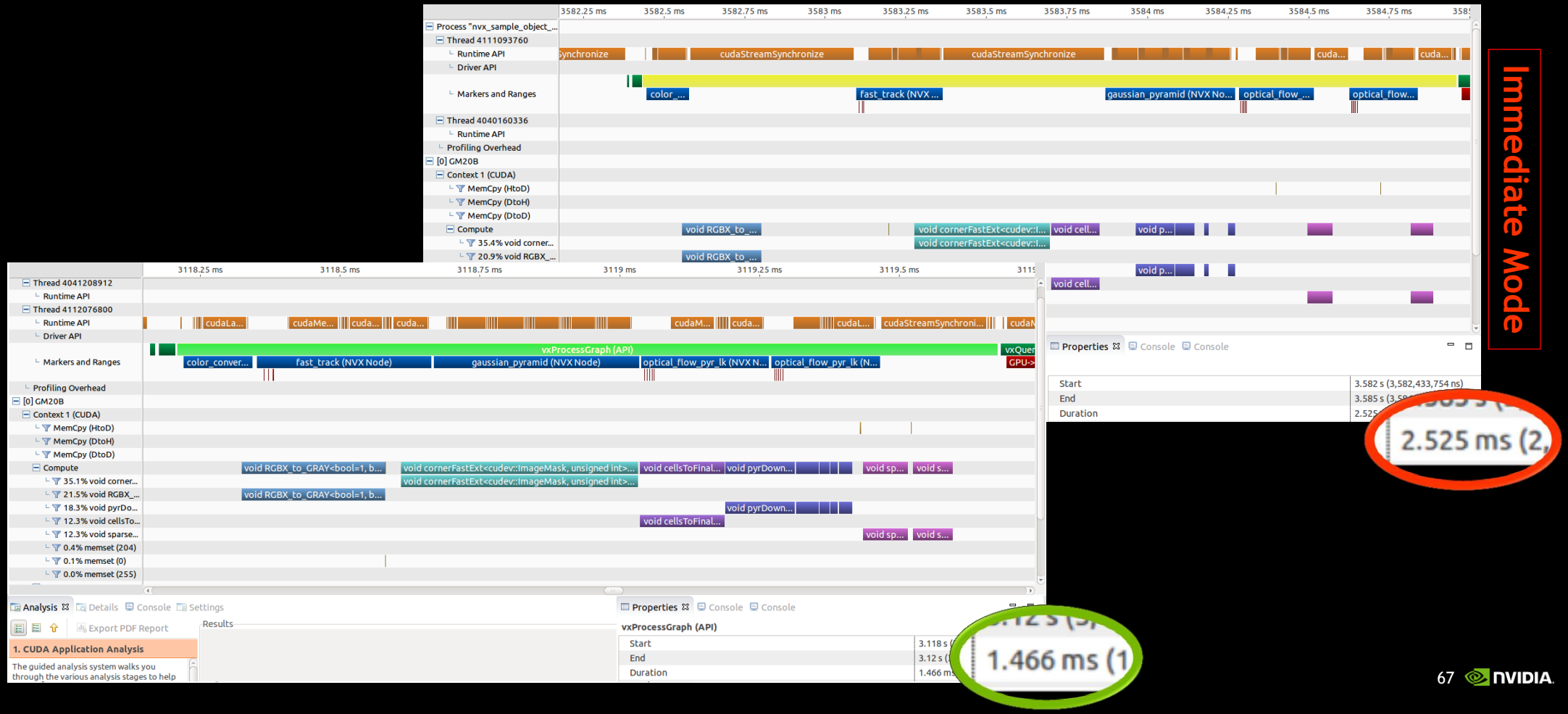


ENSURE BEST PERFORMANCE

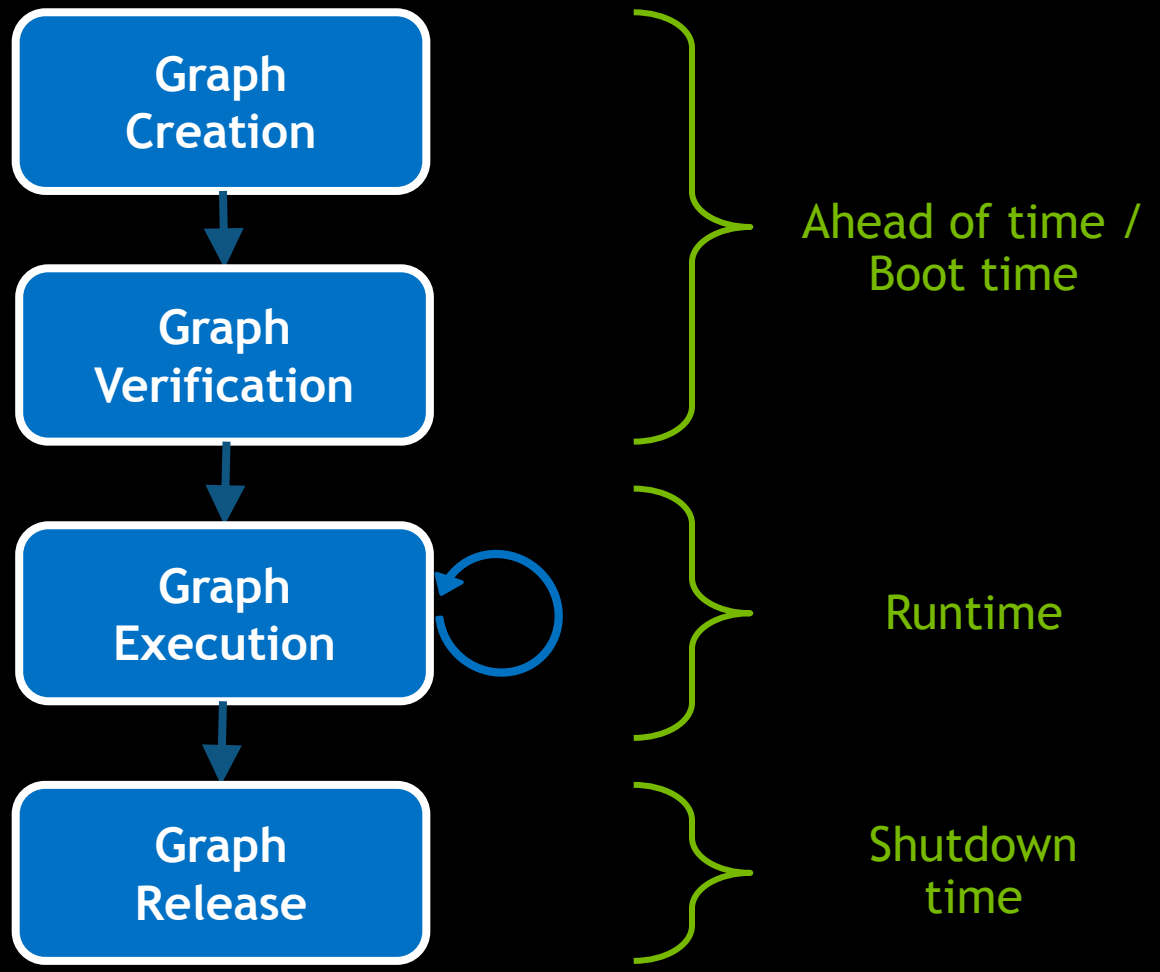
NVIDIA VISIONWORKS

Immediate Mode

Graph Mode



GRAPH LIFE CYCLE



A. GRAPH CREATION

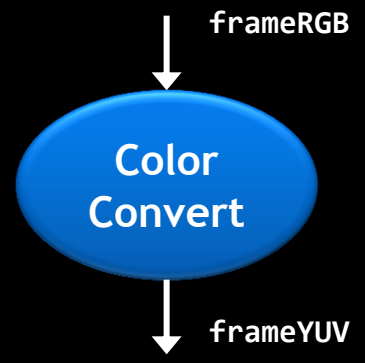
Graph and Nodes

Any number of graphs can be created

```
vx_graph graph = vxCreateGraph(context);
```

Node = instance of vision primitive with well defined parameters

```
vx_image frameRGB, frameYUV; // Already created
vx_node color = vxColorConvertNode(graph, frameRGB, frameYUV);
```



A. GRAPH CREATION

Edges

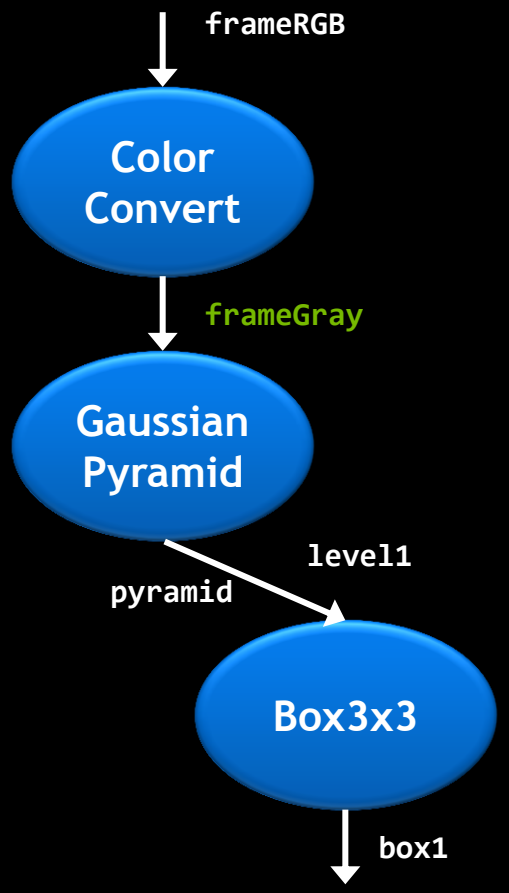
Implicit (no edge object)

Determined from nodes parameters (write → read relationships)

```
vxColorConvertNode(graph, frameRGB, frameGray);
vxGaussianPyramidNode(graph, frameGray, pyramid);
```

Object hierarchy considered in the dependency analysis (pyramid levels, ROIs)

```
vx_image level1 = vxGetPyramidLevel(pyramid, 1);
vxBox3x3Node(graph, level1, box1);
```



B. GRAPH VERIFICATION

What it does ?

Error checking (parameter consistency, graph cycles)

Memory allocation

Node or device related initialization

Optimizations

Better done at setup time

B. GRAPH VERIFICATION

Must be done before execution

Explicitly by the application (preferred)

```
vx_status status = vxVerifyGraph(graph);
```

Implicitly by VisionWorks when needed

```
vx_status status = vxProcessGraph(graph); // The graph will be automatically
                                          // verified if needed at that time
```

C. GRAPH EXECUTION

Two Modes

Synchronous

```
vx_status status = vxProcessGraph(graph);
```

Asynchronous

```
vx_status status = vxScheduleGraph(graph);
// Do something on CPU
status = vxWaitGraph(graph);
```

Accessing objects used by the graph concurrently to the execution is forbidden

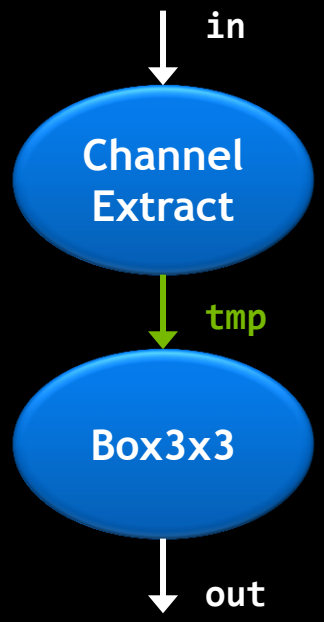
VIRTUAL DATA OBJECTS

Intermediate Graph Data

The application not allowed to access the object content

Images, arrays and pyramids can be virtual

```
vx_image in = vxCreateImage(context, 1920, 1080, VX_DF_IMAGE_NV12);
vx_image out = vxCreateImage(context, 1920, 1080, VX_DF_IMAGE_U8);
vx_image tmp = vxCreateVirtualImage(context, 1920, 1080, VX_DF_IMAGE_U8);

// Create, verify the graph and execute the graph
vx_node nExtract = vxChannelExtract(graph, in, VX_CHANNEL_Y, tmp);
vx_node nBox = vxBox3x3Node(graph, tmp, out);
```



Enables more optimizations (example : kernel fusion)

AGENDA

Programming Basics

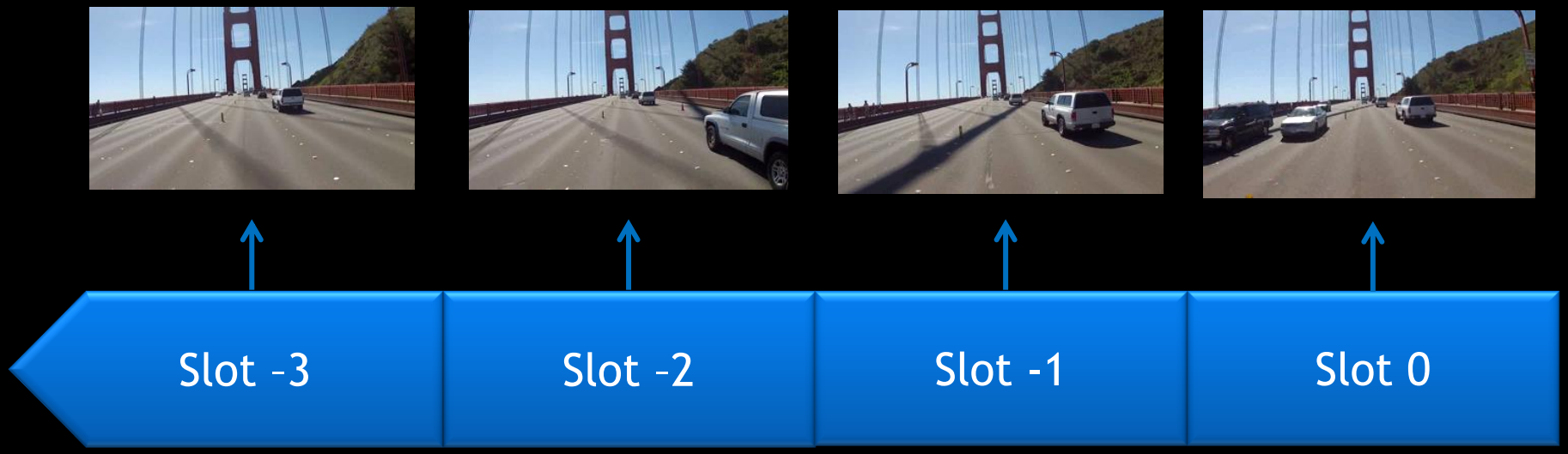
Efficient IO

Graph and Delay

- Graphs
- Delay Object
- Code Example

DELAY

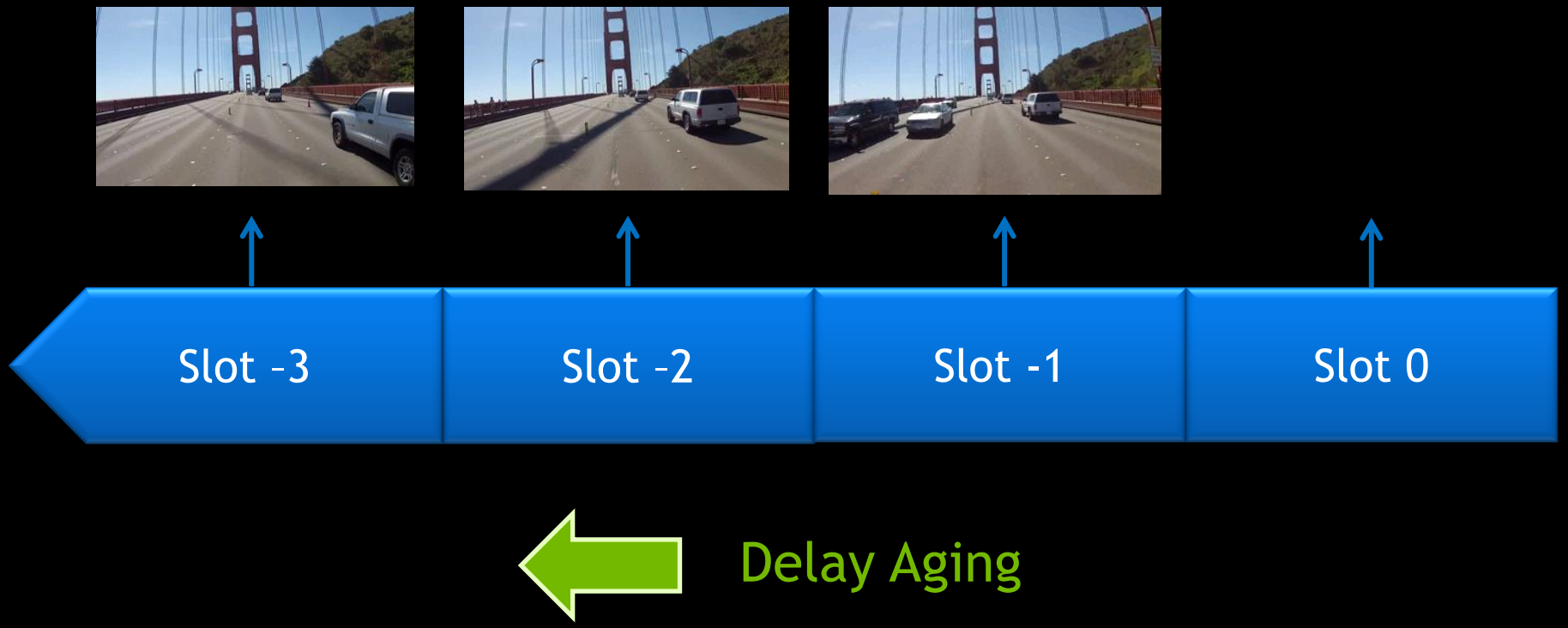
Collection of Data Objects



4 slots delay containing images

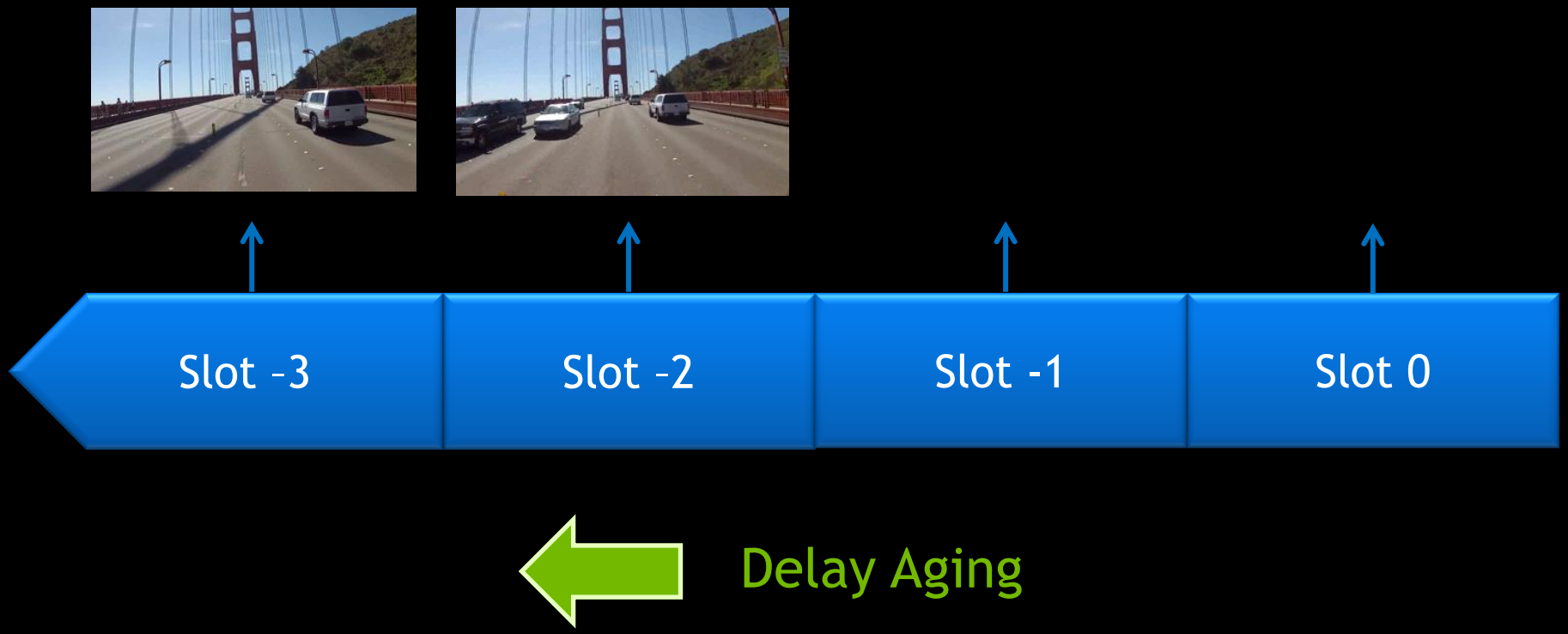
DELAY

Rolling Buffer



DELAY

Rolling Buffer



DELAY API

Creation

Exemplar object replicated in each delay slot (meta-data)

```
vx_image exemplar = vxCreateImage(context, 640, 480, VX_DF_IMAGE_RGB);

// Create a 3 slot delay containing 640x480 VX_DF_IMAGE_RGB images
vx_delay delay = vxCreateDelay(context, (vx_reference)exemplar, 3);

// The exemplar can now be released immediately
vxReleaseImage(&exemplar);
```

The exemplar object:

- Can be any data object (image, array, pyramid, …)
- Not affected by the delay creation, not link to the delay
- No memory allocation if created and released just for the delay

DELAY API

Get a Reference of Objects in Slots

The delay slot can be used as any object

```
// Get references to delay slots
vx_image prev_img = (vx_image)vxGetReferenceFromDelay(delay, -1);
vx_image cur_img = (vx_image)vxGetReferenceFromDelay(delay, 0);

// Add a node to the graph
vxAbsDiffNode(context, prev_img, cur_img, out)
```

Exception: the delay slot must not be released

DELAY API

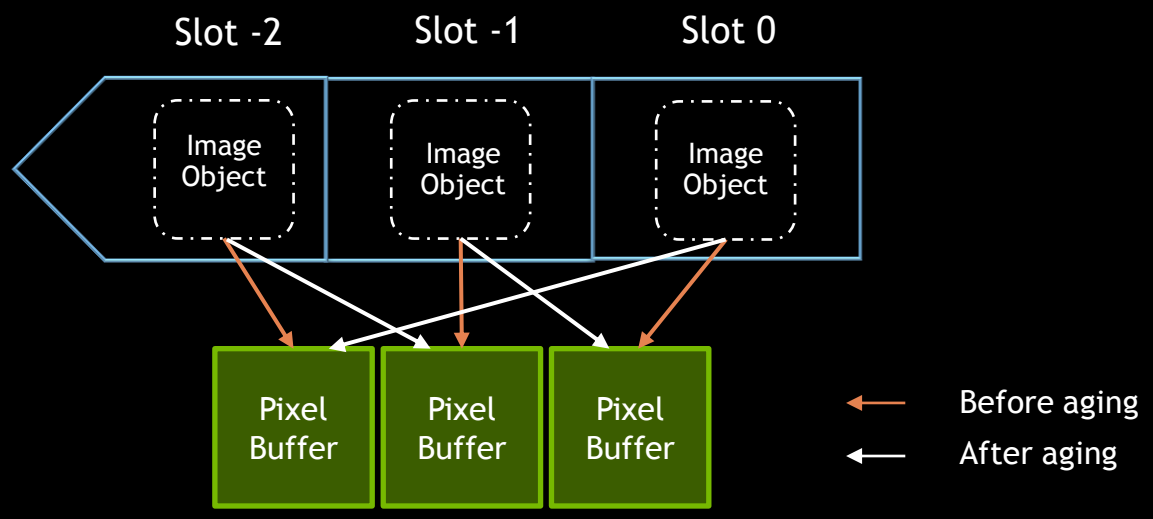
Aging

Aging a delay moves *data* from object in slot *n* to object in slot *n-1*

Only data content shift, not objects

Zero copy : internal handle switch

```
// Age the delay
vxAgeDelay(delay);
```



AGENDA

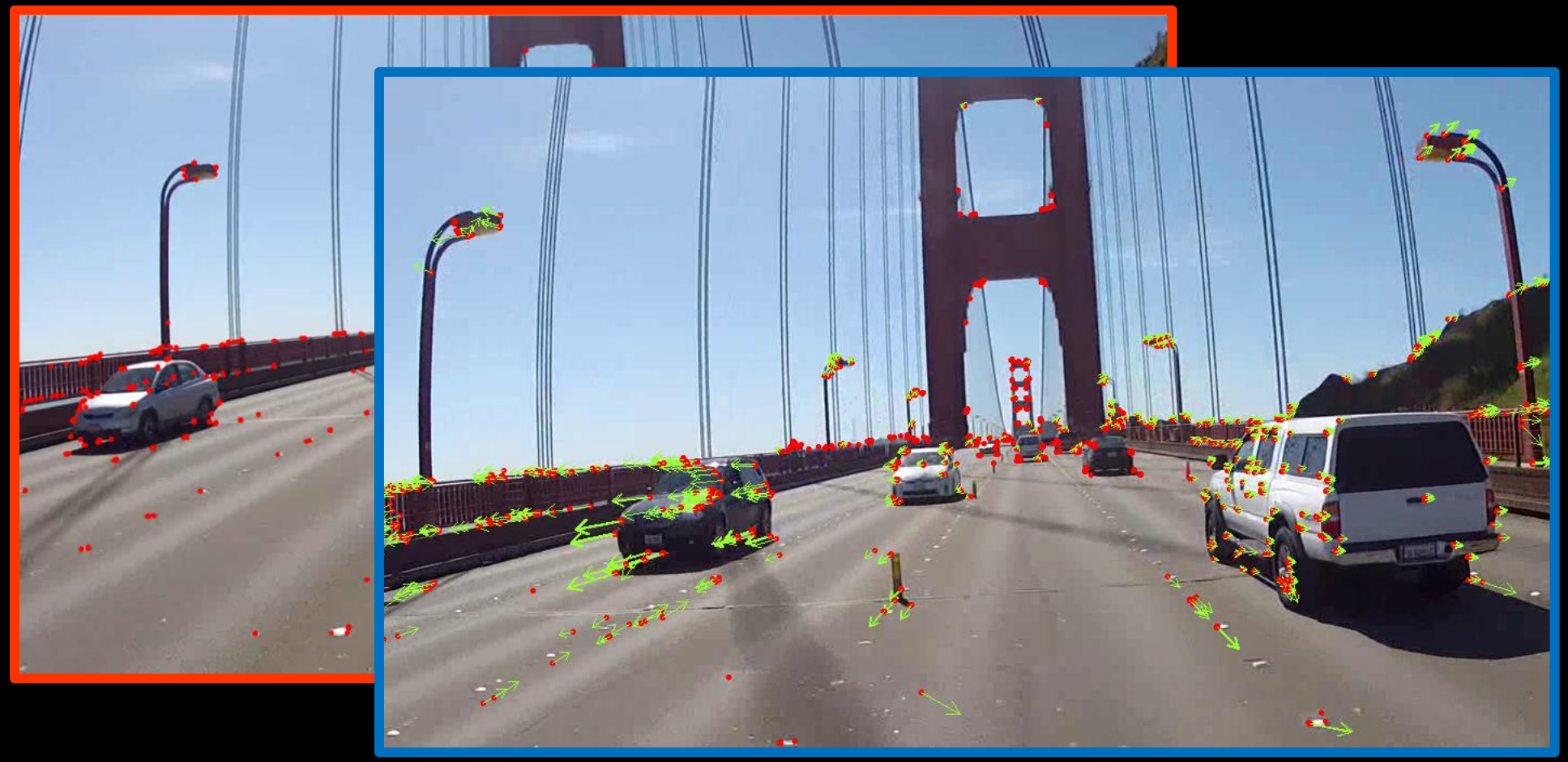
Programming Basics

Efficient IO

Graph and Delay

- Graphs
- Delay Object
- Code Example

FEATURE TRACKING



FEATURE TRACKING

What is in the Toolkit ?

Feature detector primitives

- `HarrisCorner` : strongest Harris points in the image
- `FastCorner` : strongest FAST points in the image
- `HarrisTrack` : balanced (per cell) Harris (re)detection
- `FastTrack` : balanced (per cell) FAST (re)detection

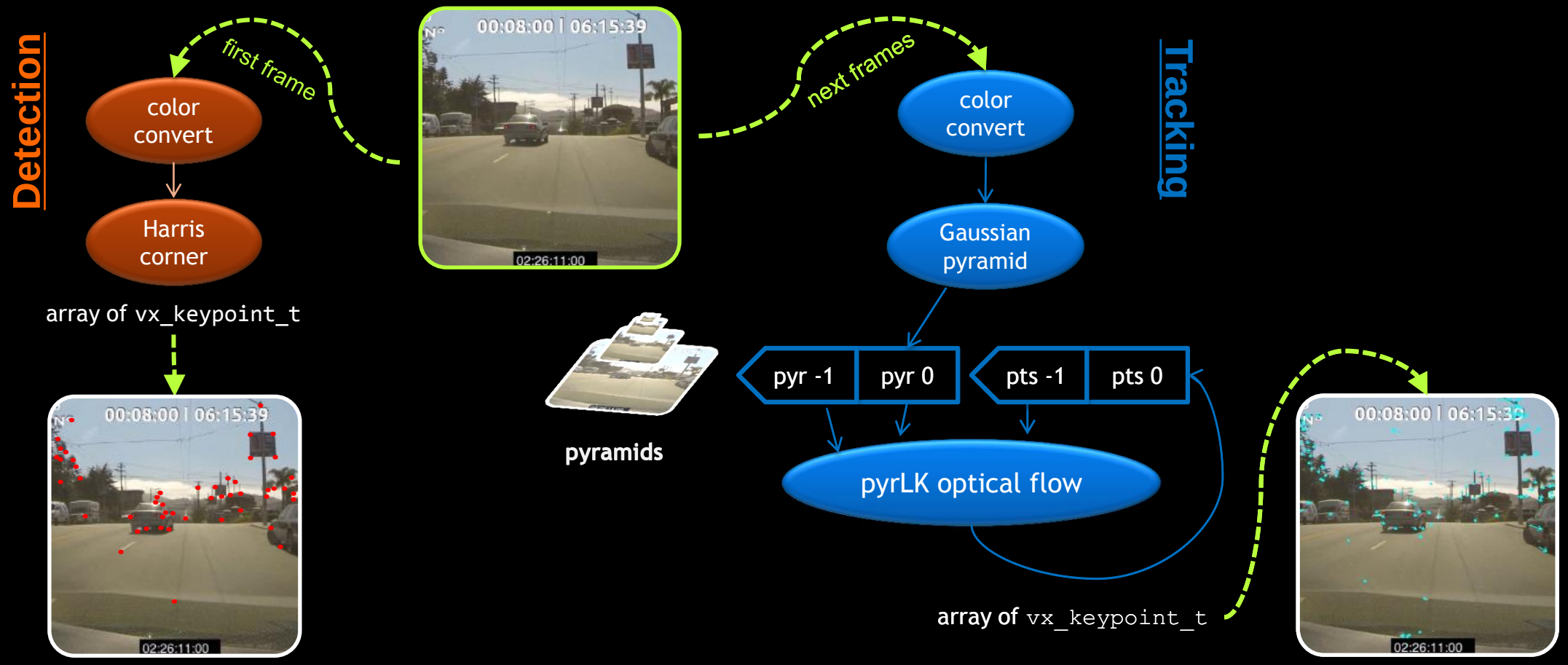
OpticalFlow / tracking primitives

- `OpticalFlowPyrLK` : sparse pyramidal Lucas-Kanade optical flow
- `GaussianPyramid` : generate a pyramid from an image

Keypoint structures

- `vx_keypoint_t` : int coordinates, strength, tracking error & status, …
- `nvx_keypoint_t` : same as `vx_keypoint_t` except float coordinates
- `nvx_point2f_t` : lightweight structure, only float coordinates

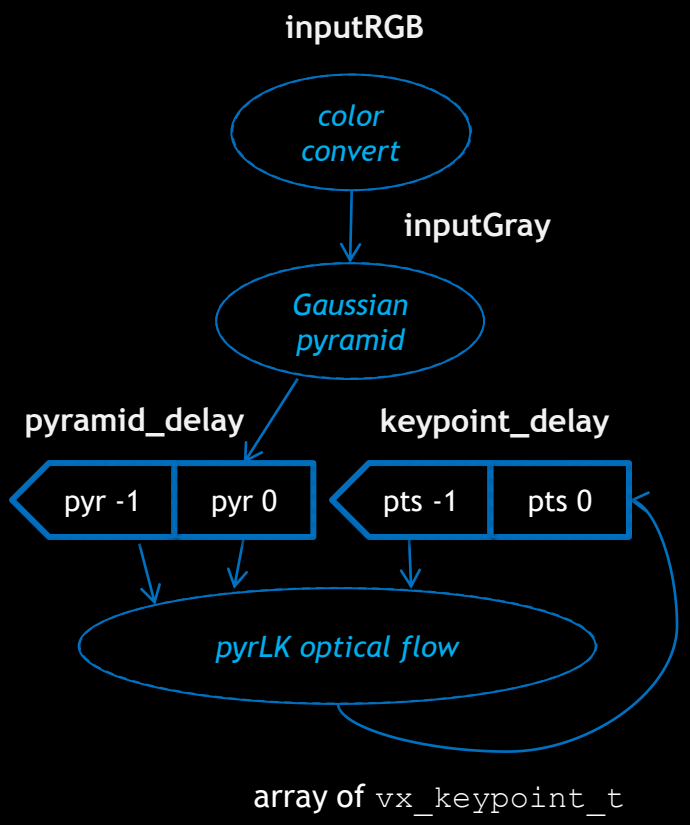
FEATURE TRACKING PROCESSING



TRACKING GRAPH

Data Objects

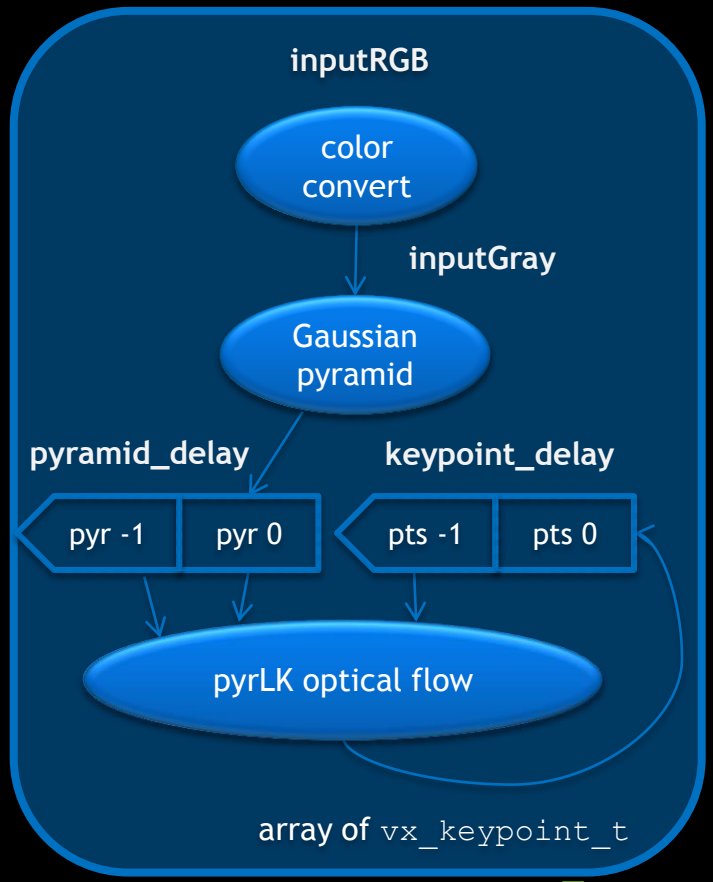
```
// Import the input data into VisionWorks
vx_image inputRGB  = vxCreateImageFromHandle(context, VX_DF_IMAGE_RGB,
                                             &addr, ptrs, NVX_IMPORT_TYPE_CUDA);

// Create the intermediate image
vx_image inputGray = vxCreateImage(context, width, height, VX_DF_IMAGE_U8);

// Image pyramids for two successive frames (2 slots delay object)
vx_pyramid pyramid_exemplar = vxCreatePyramid(context, 4, VX_SCALE_PYRAMID_HALF,
                                              width, height, VX_DF_IMAGE_U8);
vx_delay pyramid_delay = vxCreateDelay(context, (vx_reference)pyramid_exemplar, 2);
vxReleasePyramid(&pyramid_exemplar);

// Tracked points need to be stored for two successive frames (2 slot delay object)
vx_array keypoint_exemplar = vxCreateArray(context, VX_TYPE_KEYPOINT, 2000);
vx_delay keypoint_delay = vxCreateDelay(context,(vx_reference)keypoint_exemplar, 2);
vxReleaseArray(&keypoint_exemplar);
```



TRACKING GRAPH

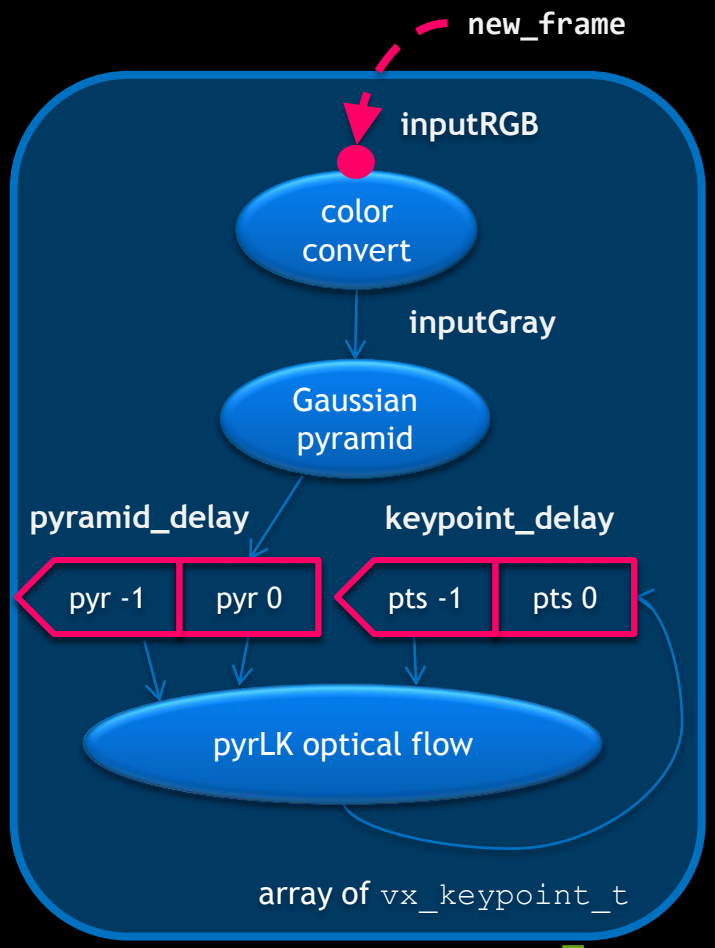
Nodes and Verification

```
vx_graph graph = vxCreateGraph (context);

// RGB to Y conversion nodes
vxColorConvertNode (graph, inputRGB, inputGray);

// Pyramid node
vx_node pyr_node = vxGaussianPyramidNode (graph, inputGray,
                          (vx_pyramid) vxGetReferenceFromDelay(pyramid_delay, 0));

// Lucas-Kanade optical flow node, previous keypoints given as new estimates
vxOpticalFlowPyrLKNode (graph,
  (vx_pyramid) vxGetReferenceFromDelay(pyramid_delay, -1), // previous pyramid
  (vx_pyramid) vxGetReferenceFromDelay(pyramid_delay, 0), // current pyramid
  (vx_array) vxGetReferenceFromDelay(keypoint_delay, -1), // previous keypoints
  (vx_array) vxGetReferenceFromDelay(keypoint_delay, -1), // new keypoints estimate
  (vx_array) vxGetReferenceFromDelay(keypoint_delay, 0),  // new keypoints
  VX_TERM_CRITERIA_BOTH, s_lk_epsilon, s_lk_num_iters,
  s_lk_use_init_est, 10);

// Graph verification
status = vxVerifyGraph (graph);
```



TRACKING EXECUTION

```
// Data objects creation
// <...>

// Graph creation & verification
// <...>

// Process the first frame (keypoints detection)
// <...>

// Main processing loop
for (;;) {
    void *devptr = 0;
    vx_rectangle_t rect = {0, width, 0, height};
    vxAccessImagePatch (inputRGB, &rect, &addr, 0, &devptr, NVX_WRITE_ONLY_CUDA);
    // Get next frame data into 'devptr' here
    // <...>
    vxCommitImagePatch (inputRGB, &rect, 0, &addr, devptr);

    // Process graph
    vxProcessGraph (graph);

    // 'Age' delay objects for the next frame processing
    vxAgeDelay (pyramid_delay);
    vxAgeDelay (keypoint_delay);
}
```



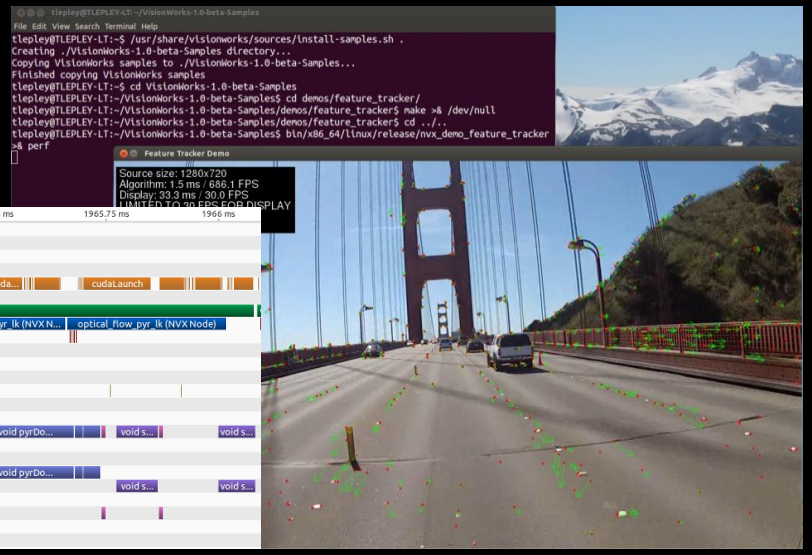
QUESTIONS ?

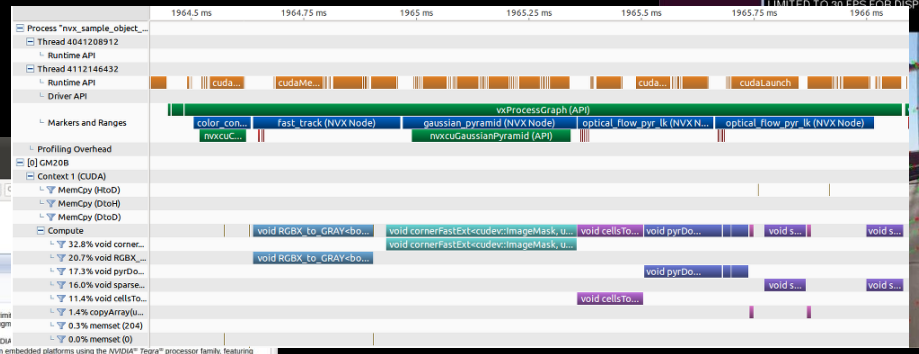
INTERESTED IN OTHER DEVELOPMENT ASPECTS ?

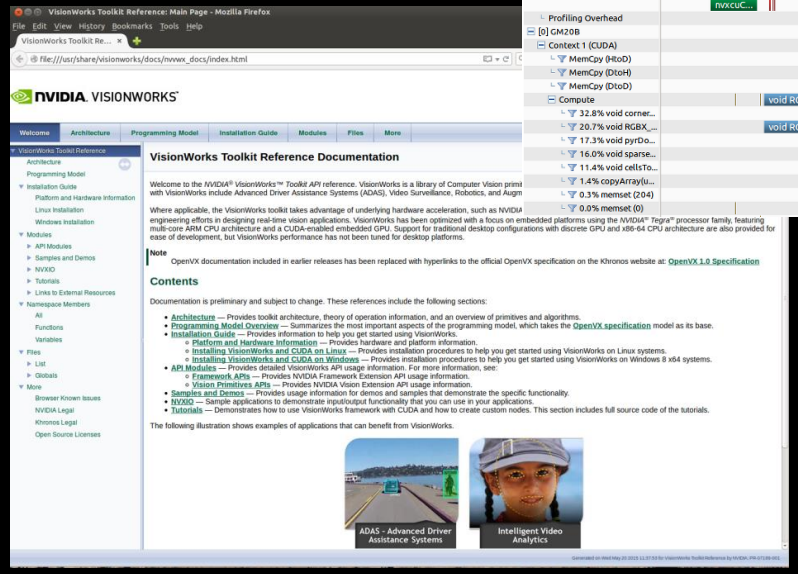
Samples

Documentation

Debug and profiling







VisionWorks Toolkit Hands-on lab (L6129)

MORE QUESTIONS ?

VisionWorks Hangout (H6115)

GPU TECHNOLOGY CONFERENCE

April 4-7, 2016 | Silicon Valley

THANK YOU

JOIN THE CONVERSATION

#GTC16

JOIN THE NVIDIA DEVELOPER PROGRAM AT developer.nvidia.com/join

PRESENTED BY

NVIDIA.